Assessing Requirements Volatility and Risk using Bayesian Networks

Michael S. Russell
Booz | Allen | Hamilton
russell_michael@bah.com

Abstract

There are many factors that affect the level of requirements volatility a system experiences over its lifecycle and the risk that volatility imparts. Improper requirements generation, undocumented user expectations, conflicting design decisions, and anticipated / unanticipated world states are representative of these volatility factors. Combined, these volatility factors can increase programmatic risk and adversely affect successful system development. This paper proposes that a Bayesian Network can be used to support reasonable judgments concerning the most likely sources and types of requirements volatility a developing system will experience prior to starting development; and by doing so it is possible to predict the level of requirements volatility the system will experience over its lifecycle. This assessment offers valuable insight to the system's developers, particularly by providing a starting point for risk mitigation planning and execution.

Introduction

When a new system is being considered for development, the system's users, developers, and other stakeholders establish a set of requirements to be implemented by the system. These requirements run the gamut from high-level concepts to design-level implementation. Over time, these requirements change as the system concept matures, user needs change, technology advances, or in response to a host of other factors (Armour, 2000). Requirements volatility is one term that describes this change. Volatility, the inverse of stability, is not necessary a bad thing. Some program managers would like to have a volatile schedule – as long as it is always being extended. However, changing requirements is generally viewed as detrimental to the program.

Requirements volatility makes its presence known in development projects of all sizes and types. Jones (Jones, 1994) noted that more than 70% of large software application development programs experience volatility; and this volatility, combined with poor requirements development processes and inadequate risk management, contributes to poor system quality, schedule slips and cost overruns. Jones also found that of the 60 projects surveyed, over 35% experienced scope or purpose related requirements volatility. A recent analysis of 44 different system development efforts (Stark, 2002) found that volatility affected about 63% of the system's initial requirements, and 6% of these directly impacted the system's scope. Additionally, it was found that requirement additions were much more likely than requirement deletions during the development phase of the lifecycle. Additionally, Stark found that customer changes accounted for 36% of overall volatility, system requirement developers accounted for 28%, and 36% were attributable to the system's developers.

Obviously, quantifying the effect of requirements volatility on a system would

benefit program managers and other stakeholders most notably supporting the identification system development risks. If the stakeholders were able to look at each factor that causes volatility and derive a quantifiable indicator of its impact prior to program initiation, they would have the information needed to plan for the mitigation or avoidance of each volatility factor. Additionally, process standards such as Carnegie Mellon's CMMI[SM] (CMMI, 2001) require programs to track and assess the impact of requirements volatility as part of their program metrics. This paper proposes a method for identifying volatility factors, assessing them, and providing useful information to the decision maker concerning the likely impact volatility will have on the program.

Methods to quantify volatility have been previously proposed. Quality Goal Modeling (Myers, 1988) is a rules-based approach designed for software developers using software quality metrics to validate requirements and to identify potentially volatile requirements. Quality Goal Modeling judges the relative volatility risk of each system requirement in terms of imprecision, conflict, and multiplicity. Additionally, it presupposes the program manager is knowledgeable enough of new system to list its essential characteristics and rank those characteristics in importance. This ranking is essential to determining the impact each requirement may have should it change. The method proposed in the paper used system characteristics as a volatility risk indicator, employing Bayesian analysis rather than rules based analysis. Using Bayesian analysis results in a more scaleable analysis approach that can benefit from volatility analysis preformed in previous programs.

York (York, 2001) proposed the Volatility of Requirements Assessment Method (VRAM) to uncover potentially volatile requirements early in a system's lifecycle. VRAM uses the Analytic Hierarchy Process (AHP) to compare system requirements against historical causes of volatility, such as "User Needs Change." The results of this process are used as a decision aide to help program managers determine if additional requirements analysis should be conducted prior to beginning system development. York considered his research inconclusive and found that even experts were not able to accurately predict volatility. He emphasized the importance of future research, especially concerning enhanced support aids to engineers in assessing volatility. This paper proposes using a Bayesian Network to provide a decision support aid.

The first challenge to developing this method is determining the situations indicative of future requirements volatility, followed by determining the impact these situations have on the system's development. As it is impractical to quantify the entire set of situations that might impact requirements volatility, a representative set must be chosen. Then the relationship between this representative set and the anticipated level of requirements volatility must be established. Often, this relationship is expressed in terms of experience-based belief, rather than with hard data, which tends to complicate any attempt to quantify the effects of volatility.

The Bayesian approach to statistical modeling relies on prior evidence to provide a rational basis for design making (Lee, 1989), and the idea of using Bayesian Analysis as a decision tool was noted in (von Winterfeldt, 1996). Bayesian statisticians are well versed in using limited or incomplete data, unquantifiable beliefs,

and other "soft" evidence to derive useful information for decision makers. By understanding the impact specific volatility factors have had on past system development efforts the program manager can use Bayesian analysis to turn this previous information into a reasonable and defendable prediction as to the level of requirements volatility the new system may experience. Other more commonly used statistical methods rely on vast quantities of hard data to make an inference as to what the data might mean. Using Bayesian analysis allows the program manager to make a reasonable judgment about volatility early in the program when the large amounts of data needed to support other statistical methods is not available.

Requirement Volatility Factors

There are as many factors for requirements volatility as there are people who write requirements, with each person having his/her own understanding why requirements change and the effect of specific volatility factors. As it is not feasible to evaluate every potential source of requirements volatility and quantify its effect on a developing system, a representative set of volatility factors that most directly effect system development should be evaluated. Ideally, this representative set would be general enough to be domain insensitive and applicable to a wide assortment of development systems. By adhering to this ideal; the method, and any tools developed using it, could be applied to many different programs. Additionally, the lessons learned in each program can be retained and used to provide better estimates of requirements volatility in future programs.

The following set of volatility factors was derived from literature (Sommerville, 1992; Brooks, 1987; Christel 1992), interviews with program managers, systems engineers, and examinations of previous system development efforts, as appears below.

- **Schedule Stability:** Measures the anticipated stability of the project's schedule. A shorted schedule can affect requirements development through the elimination of requirements engineering time, resulting in missed or poorly specified requirements. A shorter the development schedule may mean that some requirements initially specified will have to be dropped, which affects the overall system design. While shortening a schedule could be considered to be detrimental, lengthening a schedule may not necessary be good. Sometimes a longer schedule gives the systems stakeholders more opportunities to change requirements. The effects of any schedule change should be carefully considered.
- **Budget Stability:** Measures the anticipated stability of the project's budget. Increases in project budget often come with additional, unplanned requirements. These late arriving requirements pose integration challenges. Decreases in budget may cause non-core functional requirements to be dropped, which affects overall requirements stability and integration.
- **Scope Stability**: Measures the anticipated stability of the project's scope. Changes in the project's scope may have a serious impact on the requirements defined for the system. In the worst case, the purpose for the system may be completely changed, leading to a

whole new set of system requirements.

- **System Need:** Measures the level at which the user's need for the new system has been established. Without clearly defining the user's need for the system, the requirements that are critical to making the system useful for its intended audience may not be documented and subsequently built into the system.
- **Changing Priorities:** Measures whether or not, and how often, the system's customer's priorities change. Changing priorities are related to changing needs; however, where system need deals with how that system will solve a problem the customer has priorities measure how critical that need is. A system that starts out as a high priority will be provided with plentiful resources and development time.
- **Changing Expectations:** Measures how often or to what extent the customer's expectations for the system change. Expectations are hard to quantify, as they are rarely documented and may not appear as defined requirements. Expectations not only drive how a project is perceived, but also its future success and how individuals react to it. Many times the system's customer may anticipate the system will meet a specific need while the testable requirements that would enable that expectation to be met are never documented.
- **Operational Concept Stability:** Measures the stability of the systems operational concept. The operational concept defines the system's place in the world and how it fits into the overall enterprise. It also describes how the ultimate user, who may not be the system's customer, intends to use the system to accomplish a mission resulting in a shared vision for the system (Wheatcraft, 2003).
- **System Interface Plan:** Measures whether or not a system interface document is scheduled for development. The system interface document lays out physical and functional designs for how the developing system will interface with other systems.
- **System Design Plan:** Measures whether or not a system design document is scheduled for development. Often this document represents the first time all system requirements are identified and documented (Wilson 1997). This level of detail is normally not included in system scope and operational need documents.
- **System Test Plan:** Measures whether or not a system test plan will be produced and, if so, how formal the system test process will be. Without a clearly defined system testing approach, it is impossible to know whether or not the system requirements have been met or whether the documented requirements are the right ones.
- **Technical Change:** Many systems, especially those in the information technology domain, are required to incorporate the leading edge of technology. Unfortunately, technology constantly changes, and this change is rapid (Armour, 2000). Many times a newly developed system is obsolete when fielded due to rapid technology change. Even when technology changes don't have a direct impact on system is development it may have an impact

on the customer's expectations for the system.

- **Requirement Traceability:** Measures the extent to which requirements are traceable to user needs, expectations, and the system's scope. Requirements not directly traceable to one of these or to another requirement are prime candidates to be modified or dropped.
- **Requirement Conflicts:** Measures the expected number of requirement conflicts. Conflicts can occur in many different, sometimes unanticipated places within the system's design and can be difficult to predict during system's planning stages. A conflict, such as a messaging protocol not matching the communications network that it must be transmitted on, must be adjudicated with the customer prior to system design finalization. There are many requirements engineering software tools on the market, such as DOORs®, that can be used to support the projects requirement development effort. By implementing these tools, the number of conflicting requirements is generally reduced; and traceability between requirements and from requirements to systems concepts and objectives is increased.
- **Implied Requirements:** Measures the expected number of implied requirements. An implied requirement is not specifically stated by the customer, but must be implemented in order to realize the customer's original requirement. One root cause of implied requirements is unstructured, natural language in requirements development which leads to ambiguity, inaccuracy, and assumed requirements (Stokes, 1991). Additionally, what seems like a simple requirement or requirement change to the customer brings with it costly implied requirements. For instance, adding an additional antenna to an aircraft seems like a minor change; however, any new antenna would mandate a new hole in the aircraft's pressure hull requiring extensive FAA mandated pressure testing and hull recertification. These tests are time intensive and more expensive than the antenna itself.
- **Interoperability Requirements:** Measures the anticipated amount of interoperability requirements. These requirements may deal with external systems or system subcomponents. These requirements are implemented and given structure by the system interface plan. Without this plan, these requirements lack context, and potential overlaps or conflicts between them are hard to uncover. Changes to systems that must interoperate with the new system can greatly impact requirements in unanticipated ways. When evaluating the potential impact of interoperability requirements, both their use within the system and as conduits to other systems must be considered.
- **Environment Change:** Measures the impact a change in the physical environment that the system operates in may have on requirements. This can work in two ways. First, the user's need may now require the system to work in an Arctic or other extreme environment. Second, the expected environment may change in some manner.

- **System Complexity:** Measures the relative complexity of the system. This measure can vary depending on the domain. An assembly line for coat hangers isn't complex from a technology perspective; however, developing this system requires close synchronization of many mechanical subsystems. Conversely, an embedded operating system for a cellular telephone is a relativity small piece of software that represents technical complexity and reliance on interoperability standards to function correctly. Evaluating complexity as it relates to requirements volatility requires sound judgments concerning how likely the complexity of the system is to drive requirements change. In many cases, higher systems complexity is more likely to exhibit requirements volatility.
- **Reuse Requirements:** Measures the relative level of reusable component integration desired by the customer or required based on technical standards within the domain. Reusing components of existing systems, hardware and/or software, to support new system development is an increasingly common requirement. Component reuse allows for greater built-in interoperability within a domain and may lead to decreased costs, but only if interfaces to the reusable components are accessible. System complexity will increase if it is known or anticipated that the interfaces to the reuse component will be difficult to decipher.
- **Subject Matter Expert (SME) Availability:** Measures how available SMEs will be to assist in the requirements engineering process. SMEs, either from the customer or domain, are key to successfully generating stable requirements. The developer may have a good bit of domain experience; however, the best judge of how well requirements have been identified are the customer's SMEs
- **Analyst Skill:** Measures the experience level of the analysts who are working with the customer and SMEs to facilitate and document requirements.
- **Program Management Skill:** Measures the experience level of the program management team. Skill is needed in two areas to mitigate volatile requirements. First, managing the development team and the requirements development process. Second, managing the customer. The first skill is much easier to judge than the second, the second being the most vital.
- **Developer Skill:** Measures the ability of the developers to interpret and transform system requirements into system design correctly and recognize the impact that requirement change will have on the system. The developers have the best understanding of functional dependencies and can usually provide the best impact estimate for a changing requirement.
- **Defined Processes:** Measures the existence and institution of requirements engineering processes. Process standards set out a defined and repeatable process to support requirements work. When followed, they help ensure requirements are derived, documented, and changed in a reasonable manner. When an organization fails to implement a

consistent requirements engineering process, volatility will follow.

- **Project Turnover:** Measures the amount of employee turnover expected during the project. Project employee turnover can greatly affect the way requirements are documented and interpreted even with a well-defined configuration management process.
- **Customer Turnover:** Measures the amount of customer turnover expected during the project. Customer employee turnover can be a serious issue for a development team, as the new customer representative may have a completely different idea as to what the system is to do. Customer management is essential to success.
- **Company Domain Experience:** Measures the amount of experience the development organization has in the systems domain. Companies with lots of domain experience should be able to lean on that experience to produce less volatile requirements. Simply being technically able to build the product is not enough. A company that builds financial planning software may have the technical expertise to build a military command and control system, but a lack of domain knowledge will result in more volatile requirements. Of course a lack of domain experience is not always a bad thing. A developer with limited domain experience will be forced to ask many questions to fill in gaps that the customer just assumed everyone knew and didn't bother defining. With a more experienced company, these gaps are filled by implied, experienced-based requirements that may or may not meet the user's expectations.

Volatility Measurement

A note on requirements; a requirement that is not verifiable is not a requirement. In the same way, in order to make a judgment about the potential impact of a requirement volatility factor, some way to measure its impact must be established. The difficultly in measuring the impact each factor may have on overall requirements volatility is that these impacts are inherently unquantifiable. Value judgments, prior beliefs, and "gut-feelings" tend to color evaluations of factors such as "will the customers expectations change" or "how many requirement conflicts will appear."

Typical systems engineering methods, from the waterfall to the spiral, all consider volatility risk (Sommerville, 1992). In particular, the spiral was designed specifically to take volatility into account throughout the lifecycle (Boehm, 2000); however these methods do not provide a systematic method to measure potential volatility. Without a way to measure and then relate volatility factors, there will be gaps in a program's volatility analysis. Here Bayesian analysis becomes a valuable resource for judging the impact of each factor while programming planning has yet to be completed. By combining this method with the spiral development lifecycle, volatility measurement will be more rigorous and cause – effect relationships between factors will be maintained.

Building the Requirements Volatility Model

At its most fundamental level, making a judgment about how different volatility factors will affect a program during the

initial planning stages is a decision problem; one that could potentially be solved in many ways. Hopefully, the program manager will use his/her experience, or that of others, to make a reasonable and informed decision concerning the extent to which the program's requirements will be subject to change, and to characterize that change. In any case, the program manager will make assumptions about the program, its customer, and world states both within and outside his control. Based on this knowledge, the program manager can take many actions, each with its own consequences. The challenge to the program manager is to make sound judgments or inferences based on prior knowledge or the experience of others, while oftentimes not knowing all the consequences of potential actions.

In order to decide on the best actions, those that minimize or mitigate the effects of volatility on the program, the program manager must implement a method that represent s his/her beliefs about each volatility factor and make an inference about their impact to the overall program. Armed with this information, he can build volatility mitigation planning into the program plan.

Bayesian analysis is a statistical method for supporting the decision making process by representing beliefs about the world as probabilities. These probabilities are not definitive, meaning reasonable people might disagree about the validity and applicability of the resulting data. However, given informed prior information, a reasonable and defendable inference about new data based on previous data can be made (Laskey, 2003). For program managers, this means factors such as budget shortfalls and documentation problems that have caused requirements volatility on past programs can reasonably be used to predict the same problems on the current project given similar development environments.

In order to apply the Bayesian approach, a method for combining information about a project's perceived level of volatility with Bayesian reasoning must be established. This can be accomplished through the development and application of a Bayesian network. A Bayesian Network (BN), based on probability theory, is a knowledge representation that effectively captures the uncertainties and conditional independences present in a given domain. As such, it can be used to make reasonable inferences with limited data (Jenson, 1997).

BN's are drawn as directed graphs comprised of nodes and arcs. The nodes represent variables whose value is uncertain and the arcs represent dependency relationships between the variables. As a computational architecture, a BN allows the user or application to declare "evidence" on some of the nodes and, through a process called "evidence accumulation," compute revised probabilities for all other nodes in the network. (Laskey, 2002)

It has been postulated that BNs can be inadequate as a general knowledge representation language for large and complex domains (Mahoney, 1996). As noted previously, it is not reasonable to try to quantify every factor that could impact requirement volatility for a system. This is what makes the Bayesian approach so valuable. It is also important to choose volatility factors that are not domain specific, so the resulting BN can be used repeatedly and among many domains. Staying as domain generic as practical means that BN tools would not have to be customized for each project and a repository of volatility information can be created (Koller, 1997).

Unless a specialized decision support tool incorporating a Bayesian network model is available, developing this tool for a single application will consume significant resources during the beginning stages of a program. So the cost of developing the model must be weighed against the benefit the model provides to the program manager. The Bayesian network model developed to support this research provides a good starting point for program managers seeking to incorporate Bayesian inference into their decision making process.

Using BNs to help solve decision problems or derive useful information is not a panacea for every situation; however, it has been found to be very useful in solving a variety of real life problems such as quickly identifying friendly from enemy aircraft (Laskey, G, 2002). Using the volatility factors identified in the previous section as the nodes of the BN, a model was constructed using Norsys Corporation's Netica software tool.

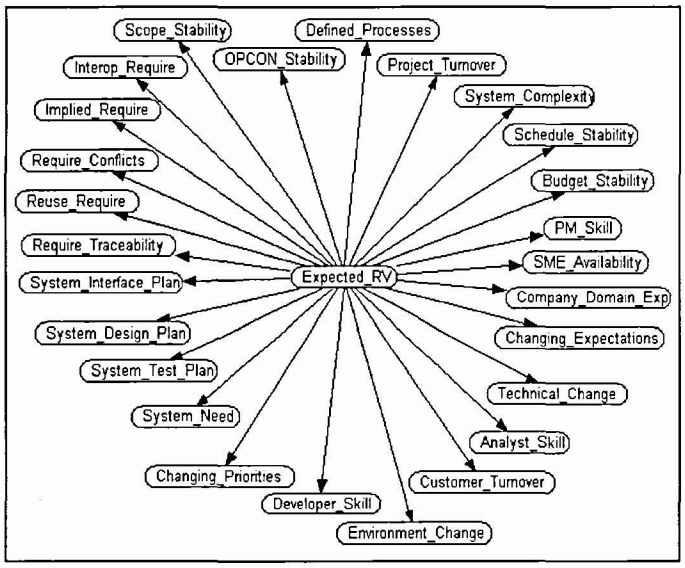

Figure 1: Requirements Volatility BN

The BN was constructed as a "Naïve Bays" Network. A naïve Bays network assumes that the features of the BN, in this case the volatility factors, are conditionally independent from each other given the expected volatility. Another type of BN is an "Optimal Classifier," which takes dependencies and other types of relationships between nodes, states, and other model elements into account. (Laskey, 2003)

The naïve BN was used for this model for two reasons. First, the optimal classifier method is more accurate; however, it carries with it a greater computational load and need for more complete information concerning the relationships between network nodes (Laskey, 2003). As the requirements volatility model is to be used during the early stages of program development and is intended to support rough order of magnitude predictions, it was felt that the data necessary to support a fully optimized model might not be present. Also, the time needed to enter and run the optimized model might limit its use by busy program managers. A method such as using a BN to predict requirements volatility must be used to be useful. This was taken into consideration. Second, research concerning interrelationships between volatility factors

is immature, and there is not general agreement within the industry as to which factors influence other factors. While some relationships are easy to infer, such as the relationship between customer expectations and a changing scope, others such as the relationship between schedule slips and requirement changes are harder to quantify. As more research in this are is completed, it would be wise to revisit the type of BN used to support the volatility model at a later date.

Developing Priors

In order to be a useful tool, the BN must be seeded with information describing the various volatility factors that have impacted systems in the past. The prior information for the requirements volatility model was developed based on reviews of pertinent literature and by surveying requirements engineers to uncover their beliefs about the root causes of requirements volatility.

The first survey was web-based and consisted of three parts. The part 1 contained 27 questions in 3 categories: (1) technical and program management skills, (2) requirement and design related, and (3) project environment and prior planning. The questions covered a range of requirements engineering and project environment challenges with the idea that each one represented a root cause of requirements volatility.

To verify that the questions asked were clearly written and appropriate to the issue at hand, a group of experienced engineers was polled to validate the survey's questions. Most possessed 10+ years of project management and requirements engineering experience. As an additional verification step, each survey respondent was asked to list the top three reasons why they felt requirements were subject to change in part 2. It was felt that part 2's "free text" entry style would be conducive to eliciting the respondent's true beliefs concerning requirements volatility. As will be noted later, the volatility factors listed in part 2 closely mirrored the more structured questioning found in part 1.

Part 3 elicited demographic information from each respondent, covering academic and work experience background as well as experience in the requirements generation process. The demographic information would be used to determine if a significant variation occurred in the answers given by respondents from different demographic groups.

Although the survey was anonymous, demographic information indicated variation among respondents covering academic, industry, and government perspectives, and encompassing a variety of experience levels. From these responses, a probability distribution for each volatility factor was developed. Based on these distributions, the relative impact of each volatility factor on overall volatility was derived.

Volatility factors dealing with the customer's expectations and defined need for the system were rated as the most likely causes of requirements volatility followed closely by instability in the system's scope. Budget stability problems, customer turnover, and technological advances within the systems domain rounded out the group of factors the survey respondents listed as the most likely causes. Based on survey responses, one of the original volatility factors dealing with the effects of an unstable Work Breakdown Structure (WBS) was removed from the model. Additionally, several factor names were changed to reflect the often-repeated responses in part 2 of the survey.

The next step in developing prior information for the BN was to conduct a second survey in which respondents were asked to assess the impact of each volatility factor on a series of five fictional scenarios. Each scenario depicted a system development project with good and bad aspects. By assessing each factor in relation to the scenario and the overall level of requirements volatility the respondent felt the system would exhibit, a matrix of responses was developed. This matrix was then used to "learn" the probability distribution of each node from the input data. Information on probabilistic learning can be found in (Laskey, 2003) and (Robert, 2001).

Using the Model to Support Decision Making

To implement the model to predict the overall level of requirements volatility, the program manager would sit down with his management team during the early stages of the program's development and record their collective beliefs as to the extent each volatility factor is present in the current program. These beliefs are entered into the model. Based on the beliefs entered for each factor (nodes within the network), the model will produce an overall measure of the requirements volatility that should be expected during the system's development.

Each volatility factor is ranked from 1 to 4. A 1 represents a factor with minimal expected impact on the system. A 4 represents a factor with a major impact, with a score of 2 or 3 being somewhere between these two extremes. As this rating scheme is inherently qualitative, the program manager must establish some ground rules for determining how these ratings should be applied to maintaining consistent results. Also, while the program manager could assign one person to make volatility judgments and complete the model, the resulting information would not be as useful as having several people with different perspectives on the program do so. By having several people work on the model, biases tend to cancel out, and a true measure of potential volatility emerges.

After each volatility factor is ranked, the data is entered into the BN tool. For the examples in this paper, the Netica tool was used. Assuming prior information was entered into the BN tool correctly; the tool will return a probability distribution that can be used to predict the level of requirements volatility the program may experience over its lifecycle. For the model in this paper, the Netica tool returns a probability distribution ranked between 1 and 10. A 1 indicates a program with a very low level of potential volatility, while a 10 represents a program with an extreme amount of potential volatility.

The prediction provided by the model is just that – a prediction. It should not be used as the sole basis for justifying risk mitigation strategies, especially expensive ones, to counteract the effects of volatile requirements. Rather, the results of the model combined with the experience of the program team work together to draw a reasonable inference and serve as a tool for mitigating potential requirements development risks. One way a program manager could use the model would be to identify the top 3-4 volatility risks and concentrate risk reduction efforts on those items. This procedure would work extremely well with a spiral development method. During spiral development, requirements will constantly shift, especially early in the lifecycle. Using the BN to predict the most likely sources of volatility and linkages between volatility factors at the

beginning of each spiral will give the program manager the information needed to begin risk reduction activities.

Conclusion:

This paper outlines one method for predicting the level of requirements volatility a system may experience during the development phase of the SE lifecycle. This prediction, with its statistical bases, provides system stakeholders with greater visibility concerning the root causes of volatility in a given program and some clue as to what portions of the system's development lifecycle are most likely to suffer from volatile requirements.

The outlined method is designed to be generic enough to be applied to many different development domains, and data captured about the impact of specific volatility factors can be reused by future programs to provide progressively better predictions of overall requirements volatility. This method also provides a high degree of flexibility to its user. By identifying additional volatility factors or inferring relationships between factors, a program manager can easily customize the BN to reflect the unique issues and other circumstances for his system.

The information used to develop the priors for the model was good enough to show the concept of using a BN for volatility prediction is sound. The next step for the method is to apply the model in a systems development environment and to judge how well the model predictions are useful to the systems stakeholders and reflect the actual level of requirements volatility the system experienced. As more information is added to the model, the model will become a better indicator of potential volatility.

References:

Armour, F., Catherwood, B., Beyers C., "A Framework for Managing Requirements Volatility using Function Points as Currency." *International Function Point Users Group Annual Conference,* San Diego, CA, September 2000.

Boehm, B., "Spiral Development, Experience, Principles, and Refinement." *Briefing to the Ballistic Missile Defense Organization,* The Pentagon, February 2000.

Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering." *IEEE Computer*, April 1987, pp 10-19.

CMMI for Systems Engineering, Software Engineering, and Integrated Product and Process Development v1.1. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 2001.

Christel, M., Kang, K., "Issues in Requirements Elicitation." *Technical Report CMU/SEI-92-TR-12*, SEI Carnegie Mellon University, Pittsburgh, PA. 1992.

Jenson, F., *An introduction to Bayesian Networks*. Springer-Verlag, NY, 1997.

Jones, C., *Assessment and Control of Software Risks*. Prentice-Hall International, Englewood Cliffs, NJ, 1994.

Koller, D., Pfeffer, A., "Object-Oriented Bayesian Networks." *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)* Providence RI, August 1997, pp 302-313.

Laskey, G., "Combat Identification with Bayesian Networks." *Student Paper*, SEOR Department, George Mason University, 2002.

Laskey, K., *Bayesian Inference and Decision Theory, SYST 644 Class Notes*. Dept. of Systems Engineering, George Mason University, http://ite.gmu.edu/~klaskey/SYST664/SYST664.html Spring 2003.

Laskey, K., Barry, P., Brouse P., *"Development of Bayesian Networks from UML Artifacts."* Dept. of Systems Engineering, George Mason University, 2002

Lee, P., *Bayesian Statistics, An Introduction.* 2nd Ed.; Oxford University Press, New York, 1989.

Mahoney S., Laskey, K., "Network engineering for complex belief networks." *Proceedings of the 12th Annual conference on Uncertainty in Artificial Intelligence. UAI-96*, 1996, pp 389-396.

Myers, M, "A Knowledge Based System for managing Software Requirements Volitility." *Doctorial Thesis, George Manson University*, Fairfax, VA., 1988

Robert, C., *The Bayesian Choice, 2nd Ed.* Springer Texts in Statistics, New York, 2001.

Solomon, P., "Managing Software Projects with Earned Value." *Northrop Grumman Corporation Internal Paper*, solompa@mail.northgrum.com, 2000.

Sommerville, I., *Software Engineering, Fourth Edition.* Addison-Wesley Publishing, Wokingham, England, 1992

orge Mason University, Fairfax, VA. 2001.

Stark, G., et al., "An examination of the Effects of Requirements Changes on Software Maintenance Releases." *To be published in the Journal of Software Maintenance*, available at http://members.aol.com/geshome/iblieve/jsmregres.pdf 2002.

Stokes, D., "Requirements Analysis." *Computer Weekly Software Engineer's Reference Book*, 1991, pp 3-21.

von Winterfeldt, D., Edwards, W., *Decision Analysis and Behavioral Research,* Cambridge University Press, Cambridge MA, 1986

Wheatcraft, L., "Delivering Quality Products that Meet Customer Expectations." *Cross Talk, VOL 16, No. 1*, January 2003, pp 11-14.

Wilson, W., "Writing Effective Requirements Specifications." *Proceedings, Software Technology Conference,* April 1997.

York, D. "An early indicator to Predict Requirements Volatility" *Doctorial Thesis, Ge*